APPENDIX A: Source Code for verification of operation of Eye-D

```
#include "main.h"

#define XDIM 512L
#define XDIMR 512
#define YDIM 480L
#define BITS 8
#define RMS_VAL 5.0
#define NUM_NOISY 16
#define NUM_DEMOS 3
#define GRAD_THRESHOLD 10

struct char_buf {
        char filename[80];
        FILE *fp;
        fpos_t fpos;
        char buf[XDIMR];
};
struct uchar_buf {
        char filename[80];
        FILE *fp;
        fpos_t fpos;
        unsigned char buf[XDIMR];
};
struct int_buf {
        char filename[80];
        FILE *fp;
        fpos_t fpos;
        int buf[XDIMR];
};
struct cortex_s {
        char filename[80];
        FILE *fp;
        fpos_t fpos;
        unsigned char buf[XDIMR];
};

struct uchar_buf test_image;
struct char_buf snow_composite;
struct uchar_buf distributed_image;
struct uchar_buf temp_image;
struct int_buf temp_wordbuffer;
struct int_buf temp_wordbuffer2;
struct uchar_buf snow_images;
struct cortex_s cortex;

int demo=0;  /* which demo is being performed, see notes */
int our_code;  /* id value embedded onto image */
int found_code=0;  /* holder for found code*/



int waitvbb(void){
        while( (_inp(PORT_BASE)&8) );
        while( !(_inp(PORT_BASE)&8) );
        return(1);
}

int grabb(void){
        waitvbb();
        _outp(PORT_BASE+1,0);
        _outp(PORT_BASE,8);
        waitvbb();
        waitvbb();
        _outp(PORT_BASE,0x10);
        return(1);
}

int livee(void){
        _outp(PORT_BASE,0x00);
        return(1);
}


int live_video(void){
        livee();
        return(1);
}

int freeze_frame(void){
        grabb();
        return(1);
}

int grab_frame(struct uchar_buf *image){
        long i;

        grabb();
        fsetpos(image->fp, &image->fpos  );
        fsetpos(cortex.fp, &cortex.fpos  );
        for(i=0;i<YDIM;i++){
                fread(cortex.buf,sizeof(unsigned char),XDIMR,cortex.fp);
                fwrite(cortex.buf,sizeof(unsigned char),XDIMR,image->fp);
        }
        livee();
        return(1);
}

int wait_vertical_blanks(int number){
        long i;
        for(i=0;i<number;i++)waitvbb();
        return(1);
}


int clear_char_image(struct char_buf *charbuffer){
        long i,j;
        char *pchar;
        fpos_t tmp_fpos;

        fsetpos(charbuffer->fp, &charbuffer->fpos  );
        for(i=0;i<YDIM;i++){
                fgetpos(charbuffer->fp, &tmp_fpos  );
                pchar = charbuffer->buf;
                fread(charbuffer->buf,sizeof(char),XDIMR,charbuffer->fp);
                for(j=0;j<XDIM;j++) *(pchar++) = 0;
                fsetpos(charbuffer->fp, &tmp_fpos  );
                fwrite(charbuffer->buf,sizeof(char),XDIMR,charbuffer->fp);
        }
        return(1);
}

int display_uchar(struct uchar_buf *image,int stretch){
        unsigned char *pimage;
        unsigned char highest = 0;
        unsigned char lowest = 255;
        long i,j;
        double dtemp,scale,dlowest;
        fpos_t tmp_fpos;

        if(stretch){
                fsetpos(image->fp, &image->fpos  );
                fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                for(i=2;i<(YDIM-2);i++){
                        fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                        pimage = &image->buf[3];
                        for(j=3;j<(XDIM-3);j++){
                                if( *pimage > highest )highest = *pimage;
                                if( *pimage < lowest )lowest = *pimage;
                                pimage++;
                        }
                }
                if(highest == lowest ){
                        printf("something wrong in contrast stretch, zero contrast");
                        exit(1);
                }
                scale = 255.0 / ( (double)highest - (double)lowest );
                dlowest = (double)lowest;
                fsetpos(image->fp, &image->fpos );
                for(i=0;i<YDIM;i++){
                        fgetpos(image->fp, &tmp_fpos );
                        fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                        pimage = image->buf;
                        for(j=0;j<XDIM;j++){
                                dtemp = ((double)*pimage - dlowest)*scale;
                                if(dtemp < 0.0)*(pimage++) = 0;
                                else if(dtemp > 255.0)*(pimage++) = 255;
                                else *(pimage++) = (unsigned char)dtemp;
                        }
                        fsetpos(image->fp, &tmp_fpos );
                        fwrite(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                }
        }


        fsetpos(image->fp, &image->fpos );
        fsetpos(cortex.fp, &cortex.fpos );
        for(i=0;i<YDIM;i++){
                fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                fwrite(image->buf,sizeof(unsigned char),XDIMR,cortex.fp);
        }
        return(1);
}


int clear_int_image(struct int_buf *wordbuffer){
        long i,j;
        int *pword;
        fpos_t tmp_fpos;

        fsetpos(wordbuffer->fp, &wordbuffer->fpos );
        for(i=0;i<YDIM;i++){
                fgetpos(wordbuffer->fp, &tmp_fpos );
                pword = wordbuffer->buf;
                fread(wordbuffer->buf,sizeof(int),XDIMR,wordbuffer->fp);
                for(j=0;j<XDIM;j++) *(pword++) = 0;
                fsetpos(wordbuffer->fp, &tmp_fpos );
                fwrite(wordbuffer->buf,sizeof(int),XDIMR,wordbuffer->fp);
        }
        return(1);
}

double find_mean_int(struct int_buf *wordbuffer){
        long i,j;
        int *pword;
        double mean=0.0;

        fsetpos(wordbuffer->fp, &wordbuffer->fpos  );
        for(i=0;i<YDIM;i++){
                pword = wordbuffer->buf;
                fread(wordbuffer->buf,sizeof(int),XDIMR,wordbuffer->fp);
                for(j=0;j<XDIM;j++) mean += (double) *(pword++);
        }
        mean /= ((double)XDIM * (double)YDIM);

        return(mean);
}

int add_uchar_to_int(struct uchar_buf *image,struct int_buf *word){
        unsigned char *pimage;
        int *pword;
        long i,j;
        fpos_t tmp_fpos;

        fsetpos(image->fp, &image->fpos  );
        fsetpos(word->fp, &word->fpos  );
        for(i=0;i<YDIM;i++){
                pword = word->buf;
                fgetpos(word->fp, &tmp_fpos  );
                fread(word->buf,sizeof(int),XDIMR,word->fp);
                pimage = image->buf;
                fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                for(j=0;j<XDIM;j++) *(pword++) += (int)*(pimage++);
                fsetpos(word->fp, &tmp_fpos  );
                fwrite(word->buf,sizeof(int),XDIMR,word->fp);
        }
        return(1);
}



int     add_char_to_uchar_creating_uchar(struct char_buf *cimage,
        struct uchar_buf *image,
        struct uchar_buf *out_image){
        unsigned char *pimage,*pout_image;
        char *pcimage;
        int temp;
        long i,j;

        fsetpos(image->fp, &image->fpos );
        fsetpos(out_image->fp, &out_image->fpos );
        fsetpos(cimage->fp, &cimage->fpos );
        for(i=0;i<YDIM;i++){
                pcimage = cimage->buf;
                fread(cimage->buf,sizeof(char),XDIMR,cimage->fp);
                pimage = image->buf;
                fread(image->buf,sizeof(unsigned char),XDIMR,image->fp);
                pout_image = out_image->buf;
                for(j=0;j<XDIM;j++){
                        temp = (int) *(pimage++) + (int) *(pcimage++);
                        if(temp<0)temp = 0;
                        else if(temp > 255)temp = 255;
                        *(pout_image++) = (unsigned char)temp;
                }
                fwrite(out_image->buf,sizeof(unsigned char),XDIMR,out_image->fp);
        }
        return(1);
}


int     copy_int_to_int(struct int_buf *word2,struct int_buf *word){
        long i;

        fsetpos(word2->fp, &word2->fpos );
        fsetpos(word->fp, &word->fpos );
        for(i=0;i<YDIM;i++){
                fread(word->buf,sizeof(int),XDIMR,word->fp);
                fwrite(word->buf,sizeof(int),XDIMR,word2->fp);
        }
        return(1);
}


void get_snow_images(void){
        unsigned char *psnow,*ptemp;
   int number_snow_inputs;
        int temp,*pword,*pword2,bit;
   long i, j;
        double rms,dtemp;

        live_video(); /* device specific */

        printf("\n\nPlease point camera at a medium lit blank wall. ");
        printf("\nDefocus the lens a bit as well ");
        printf("\nIf possible, place the camera into its highest gain, and ");
        printf("\nput the gamma to 1.0.");
        printf("  Ensure that the video is not saturated ");
        printf("\nPress any key when ready... ");

        while( !kbhit() );
        printf("\nNow finding difference frame rms value... ");

        /* subtract one image from another, find the rms difference */
        live_video();
        wait_vertical_blanks(2);
        grab_frame(&temp_image);
        live_video();
        wait_vertical_blanks(2);
        grab_frame(&distributed_image); /* use first image as buffer */

        rms = 0.0;
        fsetpos(temp_image.fp, &temp_image.fpos  );
        fsetpos(distributed_image.fp, &distributed_image.fpos  );
        for(i=0;i<YDIM;i++){
                ptemp = temp_image.buf;
                fread(temp_image.buf,sizeof(unsigned char),XDIMR,temp_image.fp);
                psnow = distributed_image.buf;
                fread(distributed_image.buf,sizeof(unsigned
char),XDIMR,distributed_image.fp);
                for(j=0;j<XDIM;j++){
                        temp = (int) *(psnow++)  -  (int) *(ptemp++);
                        dtemp = (double)temp;
                        dtemp *= dtemp;
                        rms += dtemp;
                }
        }
        rms /= ( (double)XDIM * (double)YDIM );
        rms = sqrt(rms);
        printf("\n\nAn rms frame difference noise value of %lf was found.",rms);
        printf("\nWe want at least %lf for good measure",RMS_VAL);
        /* we want rms to be at least RMS_VAL DN, so ...   */
        if(rms > RMS_VAL) number_snow_inputs = 1;
        else {
                dtemp = RMS_VAL / rms;
                dtemp *= dtemp;
                number_snow_inputs = 1 + (int)dtemp;
        }
        printf("\n%d images will achieve this noise level",number_snow_inputs);

        /* now create each snowy image */

        printf("\nStarting to create snow pictures... \n");
        fsetpos(snow_images.fp, &snow_images.fpos  ); /* set on first image*/
        for(bit = 0; bit < BITS; bit++){

                clear_int_image(&temp_wordbuffer);
                for(i=0;i<number_snow_inputs;i++){
                        live_video();
                        wait_vertical_blanks(2);
                        grab_frame(&temp_image);
                        add_uchar_to_int(&temp_image,&temp_wordbuffer);
                }

                clear_int_image(&temp_wordbuffer2);
                for(i=0;i<number_snow_inputs;i++){
                        live_video();
                        wait_vertical_blanks(2);
                        grab_frame(&temp_image);
                        add_uchar_to_int(&temp_image,&temp_wordbuffer2);
                }

                /* now load snow_images[bit] with the difference frame biased by
                128 in an unsigned char form just to keep things clean */
                /* display it on cortex also */
                fsetpos(temp_wordbuffer2.fp, &temp_wordbuffer2.fpos  );
                fsetpos(temp_wordbuffer.fp, &temp_wordbuffer.fpos  );
                fsetpos(temp_image.fp, &temp_image.fpos  );
                for(i=0;i<YDIM;i++){
                        pword = temp_wordbuffer.buf;
                        fread(temp_wordbuffer.buf,sizeof(int),XDIMR,temp_wordbuffer.fp);
                        pword2 = temp_wordbuffer2.buf;

fread(temp_wordbuffer2.buf,sizeof(int),XDIMR,temp_wordbuffer2.fp);
                        psnow = snow_images.buf;
                        ptemp = temp_image.buf;
                        for(j=0;j<XDIM;j++) {
                                *(psnow++) = *(ptemp++) = (unsigned char)
(*(pword++) - *(pword2++) + 128);
                        }
                        fwrite(snow_images.buf,sizeof(unsigned
char),XDIMR,snow_images.fp);
                        fwrite(temp_image.buf,sizeof(unsigned
char),XDIMR,temp_image.fp);
                }
                freeze_frame();
                display_uchar(&temp_image,0); /*1 signifies to stretch the contrast*/
                printf("\rDone snowy %d   ",bit);
                wait_vertical_blanks(30);
        }
        return;
}




void loop_visual(void){
        unsigned char *psnow;
        char *pcomp;
        long i,j,count = 0;
        int ok=0,temp,bit,add_it;
        double scale = 1.0 / RMS_VAL;
        double dtemp,tmpscale;
        fpos_t tmp_fpos;

        /* initial rms of each snowy image should be around 5 to 10 DN.
        let's assume it is 5, and assume further that our acceptable noise level of
        the full snowy composite should be approximately 1 DN, thus we need to
        scale them down by approximately 5*BITS as a first guess, then do the
        visual loop to zoom in on final acceptable value */


        printf("\n\n Now calculating initial guess at amplitude...  \n");
        while( !ok ){
                /* calculate snow_composite */
                /* clear composite */
                clear_char_image(&snow_composite);

                fsetpos(snow_images.fp, &snow_images.fpos  ); /* set on first image*/
                for(bit=0;bit<BITS;bit+ +){
                        j = 128 > > bit;
                        if( our_code & j)add_it=1;
                        else add_it=0;
                        fsetpos(snow_composite.fp, &snow_composite.fpos  );
                        for(i=0;i<YDIM;i+ +){
                                psnow = snow_images.buf;
                                fread(snow_images.buf,sizeof(unsigned
char),XDIMR,snow_images.fp);
                                fgetpos(snow_composite.fp, &tmp_fpos  );

fread(snow_composite.buf,sizeof(char),XDIMR,snow_composite.fp);
                                pcomp = snow_composite.buf;
                                for(j=0;j<XDIM;j+ +) {
                                        dtemp = ((double)*(psnow+ +) -128.0) * scale;
                                        if(dtemp<0.0){
                                                temp = -(int) fabs( -dtemp +0.5);
```

```
                    }
                    else {
                        temp = (int) fabs( dtemp +0.5);
                    }
                    if(temp > 127) {
                        temp = 127;
                    }
                    else if(temp < -128) {
                        temp = -128;
                    }
                    if(add_it){
                        *(pcomp++) += (char)temp;
                    }
                    else {
                        *(pcomp++) -= (char)temp;
                    }
                }
                fsetpos(snow_composite.fp, &tmp_fpos  );

fwrite(snow_composite.buf,sizeof(char),XDIMR,snow_composite.fp);
            }
            printf("\rDone snowy %d   ",bit);
        }

        /* add snow composite to test image to form dist image */
        add_char_to_uchar_creating_uchar(
            &snow_composite,
            &test_image,
            &distributed_image);

        /* display both and cue for putting scale down, up or ok */
        i=count = 0;
        printf("\n Depress any key to toggle, enter to move on...\n  ");
        printf("\r Distributed Image...     ");
        display_uchar(&distributed_image,0);
        while( getch() != '\r' ){
            if( (count++) % 2){
                printf("\r Distributed Image...     ");
                display_uchar(&distributed_image,0);
            }
            else {
                printf("\r Original Image...     ");
                display_uchar(&test_image,0);
            }
        }
        printf("\nScale = %lf ",scale);
        printf("\nEnter new scale, or >1e6 for ok... ");
        scanf("%lf",&tmpscale);
```

```
                if(tmpscale > 1e6)ok=1;
                else scale = tmpscale;
        }
        /* distributed image now is ok; calculate actual snow_images used and
        store in those arrays; */

        fsetpos(snow_images.fp, &snow_images.fpos ); /* set on first image*/
        printf("\nNow storing snow images as used... \n");
        for(bit=0;bit<BITS;bit++){
                for(i=0;i<YDIM;i++){
                        psnow = snow_images.buf;
                        fgetpos(snow_images.fp, &tmp_fpos );
                        fread(snow_images.buf,sizeof(unsigned
char),XDIMR,snow_images.fp);
                        for(j=0;j<XDIM;j++) {
                                dtemp = ((double)*psnow -128.0) * scale;
                                if(dtemp<0.0){
                                        temp = -(int) fabs( -dtemp +0.5);
                                }
                                else {
                                        temp = (int) fabs( dtemp +0.5);
                                }
                                *(psnow++) = (unsigned char)(temp + 128);
                        }
                        fsetpos(snow_images.fp, &tmp_fpos );
                        fwrite(snow_images.buf,sizeof(unsigned
char),XDIMR,snow_images.fp);
                }
                printf("\rDone snowy %d    ",bit);
        }
        return;
}




double find_grad(struct int_buf *image,int load_buffer2){
        int buf1[XDIMR],buf2[XDIMR],buf3[XDIMR];
        int *pbuf1,*pbuf2,*pbuf3,*p2;
        double total=0.0,dtemp;
   long i, j;
        fpos_t tmp_pos;

        fsetpos(image->fp, &image->fpos );
        fgetpos(image->fp, &tmp_pos );
```

```
fsetpos(temp_wordbuffer2.fp, &temp_wordbuffer2.fpos );

for(i=1;i<(YDIM-1);i++){
        fsetpos(image->fp, &tmp_pos );
        fread(buf1,sizeof(int),XDIMR,image->fp);
        fgetpos(image->fp, &tmp_pos );
        fread(buf2,sizeof(int),XDIMR,image->fp);
        fread(buf3,sizeof(int),XDIMR,image->fp);
        pbuf1=buf1;
        pbuf2=buf2;
        pbuf3=buf3;
        p2 = temp_wordbuffer2.buf;

if(load_buffer2){
        for(j=1;j<(XDIM-1);j++){
                dtemp = (double)*(pbuf1++);
                dtemp += (double)*(pbuf1++);
                dtemp += (double)*(pbuf1--);
                dtemp += (double)*(pbuf2++);
                dtemp -= (8.0 * (double) *(pbuf2++));
                dtemp += (double)*(pbuf2--);
                dtemp += (double)*(pbuf3++);
                dtemp += (double)*(pbuf3++);
                dtemp += (double)*(pbuf3--);
                *p2 = (int)dtemp;
                if( *p2 > GRAD_THRESHOLD ){
                        *(p2++) -= GRAD_THRESHOLD;
                }
                else if( *p2 < -GRAD_THRESHOLD ){
                        *(p2++) += GRAD_THRESHOLD;
                }
                else {
                        *(p2++) = 0;
                }
        }
        fwrite(temp_wordbuffer2.buf,sizeof(int),XDIMR,temp_wordbuffer2.fp);
}
else {
        fread(temp_wordbuffer2.buf,sizeof(int),XDIMR,temp_wordbuffer2.fp);
        for(j=1;j<(XDIM-1);j++){
                dtemp = (double)*(pbuf1++);
                dtemp += (double)*(pbuf1++);
                dtemp += (double)*(pbuf1--);
                dtemp += (double)*(pbuf2++);
                dtemp -= (8.0 * (double) *(pbuf2++));
                dtemp += (double)*(pbuf2--);
                dtemp += (double)*(pbuf3++);
                dtemp += (double)*(pbuf3++);
```

```
                        dtemp += (double)*(pbuf3--);

                        dtemp -= (double) *(p2++);

                        dtemp *= dtemp;
                        total += dtemp;
                }
        }
        }

        return(total);
}



void search_1(struct uchar_buf *suspect){
        unsigned char *psuspect,*psnow;
        int bit,*pword,temp;
    long i,j;
        double add_metric,subtract_metric;
        fpos_t tmp_fpos;

        /* this algorithm is conceptually the simplest.  The idea is to step
        through each bit at a time and merely see if adding or subtracting the
        individual snowy picture minimizes some 'contrast' metric.
        This should be the most crude and inefficient, no where to go but
        better */

        fsetpos(snow_images.fp, &snow_images.fpos  );
        temp=256;
        clear_int_image(&temp_wordbuffer);
        add_uchar_to_int(suspect,&temp_wordbuffer);
        find_grad(&temp_wordbuffer,1); /* 1 means load temp_wordbuffer2 */
        for(bit=0;bit<BITS;bit++){
                /* add first */
                fgetpos(snow_images.fp, &tmp_fpos  );
                fsetpos(suspect->fp, &suspect->fpos  );
                fsetpos(temp_wordbuffer.fp, &temp_wordbuffer.fpos  );
                for(i=0;i<YDIM;i++){
                        pword = temp_wordbuffer.buf;
                        psuspect = suspect->buf;
                        psnow = snow_images.buf;
                        fread(suspect->buf,sizeof(unsigned char),XDIMR,suspect->fp);
                        fread(snow_images.buf,sizeof(unsigned
char),XDIMR,snow_images.fp);
                        for(j=0;j<XDIM;j++){
                                *(pword++)=(int)*(psuspect++)+(int)*(psnow++)-128;
                        }
                        fwrite(temp_wordbuffer.buf,sizeof(int),XDIMR,temp_wordbuffer.fp);
                }
                add_metric = find_grad(&temp_wordbuffer,0);

                /* then subtract */
                fsetpos(snow_images.fp, &tmp_fpos  );
                fsetpos(suspect->fp, &suspect->fpos  );
                fsetpos(temp_wordbuffer.fp, &temp_wordbuffer.fpos  );
                for(i=0;i<YDIM;i++){
                        pword = temp_wordbuffer.buf;
                        psuspect = suspect->buf;
                        psnow = snow_images.buf;
                        fread(suspect->buf,sizeof(unsigned char),XDIMR,suspect->fp);
                        fread(snow_images.buf,sizeof(unsigned
char),XDIMR,snow_images.fp);
                        for(j=0;j<XDIM;j++){
                                *(pword++)=(int)*(psuspect++)-(int)*(psnow++)+128;
                        }
                        fwrite(temp_wordbuffer.buf,sizeof(int),XDIMR,temp_wordbuffer.fp);
                }
                subtract_metric = find_grad(&temp_wordbuffer,0);

                printf("\nbit place %d: add=%le ,
sub=%le",bit,add_metric,subtract_metric);
                temp/=2;
                if(add_metric < subtract_metric){
                        printf("  bit value = 0");
                }
                else {
                        printf("  bit value = 1");
                        found_code += temp;
                }
        }
        printf("\n\nYour magic number was %d",found_code);
        return;
}


void search_2(unsigned char *suspect){

    if(suspect);

        return;
}
```

```
void loop_simulation(void){
        unsigned char *ptemp,*pdist;
        int *pword,int_mean,ok=0,temp;
        long i,j;
        double mean,scale;

        /* grab a noisy image into one of the temp buffers */
        printf("\ngrabbing noisy frame...\n");
        clear_int_image(&temp_wordbuffer);
        for(i=0;i<NUM_NOISY;i++){
                live_video();
                wait_vertical_blanks(2);
                grab_frame(&temp_image);
                add_uchar_to_int(&temp_image,&temp_wordbuffer);
                j=(long)NUM_NOISY;
                printf("\r%ld of %ld    ",i+1,j);
        }

        /* find mean value of temp_wordbuffer */
        mean = find_mean_int(&temp_wordbuffer);
        int_mean = (int)mean;

        /* now we will add scaled version of this 'corruption' to our distributed
        image */
        scale = 1.0;
        while( !ok ){
                /* add noise to dist image storing in temp_image */
                fsetpos(distributed_image.fp, &distributed_image.fpos );
                fsetpos(temp_wordbuffer.fp, &temp_wordbuffer.fpos );
                fsetpos(temp_image.fp, &temp_image.fpos );
                for(i=0;i<YDIM;i++){
                        pdist = distributed_image.buf;
                        pword = temp_wordbuffer.buf;
                        ptemp = temp_image.buf;
                        fread(distributed_image.buf,sizeof(unsigned
char),XDIMR,distributed_image.fp);
                        fread(temp_wordbuffer.buf,sizeof(int),XDIMR,temp_wordbuffer.fp);
                        for(j=0;j<XDIM;j++){
                                temp = (int) *(pdist++) + *(pword++) - int_mean;
                                if(temp<0)temp = 0;
                                else if(temp > 255)temp = 255;
                                *(ptemp++) = (unsigned char)temp;
                        }
                        fwrite(temp_image.buf,sizeof(unsigned
char),XDIMR,temp_image.fp);
                }
                /* display the dist image and the corrupted image */
                display_uchar(&temp_image,0);

                /* apply new ’corrupted’ image to search algorithm 1 for id value */
                search_1(&temp_image);

                /* apply new ’corrupted’ image to search algorithm 2 for id value */
                /*
                search_2(temp_image);
                */

                /* prompt for upping noise content or ok */
                ok = 1;
        }

        return;
}



int initialize_everything(void){
        long i,j;
        unsigned char *pucbuf;
        char *pcbuf;
        int *pibuf;

        /* initialize cortex */
        strcpy(cortex.filename,"f:image");
        if((cortex.fp=fopen(cortex.filename,"rb"))==NULL){
                system("v f g");
        }
        else fclose(cortex.fp);
        if( (_inp(PORT_BASE) == 0xFF) ){
                printf("oops ");
                exit(0);
        }

        /* open cortex for read and write */
        if((cortex.fp=fopen(cortex.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(cortex.fp, &cortex.fpos  );

/* test_image;  original image */
        strcpy(test_image.filename,"e:tst_img");
        if((test_image.fp=fopen(test_image.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pucbuf = test_image.buf;
        for(i=0;i<XDIM;i++)*(pucbuf++)=0;
        for(i=0;i<YDIM;i++){
                fwrite(test_image.buf,sizeof(unsigned char),XDIMR,test_image.fp);
        }
        fclose(test_image.fp);
        if((test_image.fp=fopen(test_image.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(test_image.fp, &test_image.fpos  );

/* snow_composite;  ultimate image added to original image */
        strcpy(snow_composite.filename,"e:snw_cmp");
        if((snow_composite.fp=fopen(snow_composite.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pcbuf = snow_composite.buf;
        for(i=0;i<XDIM;i++)*(pcbuf++)=0;
        for(i=0;i<YDIM;i++){
                fwrite(snow_composite.buf,sizeof(char),XDIMR,snow_composite.fp);
        }
        fclose(snow_composite.fp);
        if((snow_composite.fp=fopen(snow_composite.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(snow_composite.fp, &snow_composite.fpos  );

/* distributed_image;   test_img plus snow_composite */
        strcpy(distributed_image.filename,"e:dst_img");
        if((distributed_image.fp=fopen(distributed_image.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pucbuf = distributed_image.buf;
        for(i=0;i<XDIM;i++)*(pucbuf++)=0;
        for(i=0;i<YDIM;i++){
                fwrite(distributed_image.buf,sizeof(unsigned
char),XDIMR,distributed_image.fp);
        }
        fclose(distributed_image.fp);
        if((distributed_image.fp=fopen(distributed_image.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(distributed_image.fp, &distributed_image.fpos );


/* temp_image;   buffer if needed */
        strcpy(temp_image.filename,"e:temp_img");
        if((temp_image.fp=fopen(temp_image.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pucbuf = temp_image.buf;
        for(i=0;i<XDIM;i++)*(pucbuf++)=0;
        for(i=0;i<YDIM;i++){
                fwrite(temp_image.buf,sizeof(unsigned char),XDIMR,temp_image.fp);
        }
        fclose(temp_image.fp);
        if((temp_image.fp=fopen(temp_image.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(temp_image.fp, &temp_image.fpos );

/*  temp_wordbuffer;   16 bit image buffer for averaging */
        strcpy(temp_wordbuffer.filename,"e:temp_wrd");
        if((temp_wordbuffer.fp=fopen(temp_wordbuffer.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pibuf = temp_wordbuffer.buf;
        for(i=0;i<XDIM;i++)*(pibuf++)=0;
        for(i=0;i<YDIM;i++){
                fwrite(temp_wordbuffer.buf,sizeof(int),XDIMR,temp_wordbuffer.fp);
        }
        fclose(temp_wordbuffer.fp);
        if((temp_wordbuffer.fp=fopen(temp_wordbuffer.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(temp_wordbuffer.fp, &temp_wordbuffer.fpos );

/* temp_wordbuffer2;  /* 16 bit image buffer for averaging */
        strcpy(temp_wordbuffer2.filename,"e:tmp_wrd2");
        if((temp_wordbuffer2.fp=fopen(temp_wordbuffer2.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pibuf = temp_wordbuffer2.buf;
        for(i=0;i<XDIM;i++)*(pibuf++)=0;
        for(i=0;i<YDIM;i++){
                fwrite(temp_wordbuffer2.buf,sizeof(int),XDIMR,temp_wordbuffer2.fp);
        }
        fclose(temp_wordbuffer2.fp);
        if((temp_wordbuffer2.fp=fopen(temp_wordbuffer2.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(temp_wordbuffer2.fp, &temp_wordbuffer2.fpos );

/* snow_images; BITS number of constituent snowy pictures */
        strcpy(snow_images.filename,"snw_imgs");
        if((snow_images.fp=fopen(snow_images.filename,"wb"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        pucbuf = snow_images.buf;
        for(i=0;i<XDIM;i++)*(pucbuf++)=0;
        for(j=0;j<BITS;j++){
        for(i=0;i<YDIM;i++){
                fwrite(snow_images.buf,sizeof(unsigned char),XDIMR,snow_images.fp);
        }
        }
        fclose(snow_images.fp);
        if((snow_images.fp=fopen(snow_images.filename,"rb+"))==NULL){
                printf(" No good on open file joe ");
                exit(0);
        }
        fgetpos(snow_images.fp, &snow_images.fpos );

        return(1);
}


int close_everything(void){

        fclose(test_image.fp);
        fclose(snow_composite.fp);
        fclose(distributed_image.fp);
        fclose(temp_image.fp);
        fclose(temp_wordbuffer.fp);
        fclose(temp_wordbuffer2.fp);
        fclose(snow_images.fp);

        return(1);
}
```

```
main(){
        int i,j;

        printf("\nInitializing...\n\n");
        initialize_everything();  /* device specific and global mallocs */

        live_video();

        /* prompt for which of the three demos to perform */
        while( demo < 1 || demo > NUM_DEMOS){
                printf("Which demo do you want to run?\n\n");
                printf("1: Digital Imagery and Very High End Photography Simulation\n");
                printf("2: Pre-exposed Print Paper and other Dupping\n");
                printf("3: Pre-exposed Original Film (i.e. In-Camera)\n");
                printf("\nEnter number and return: ");
                scanf("%d",&demo);
                if(demo < 1 || demo > NUM_DEMOS){
                        printf("\n eh eh  ");
                }
        }

        /* acquire test image */
        printf("\nPress any key after your test scene is ready... ");
        getch();
        grab_frame(&test_image);  /*grab_frame takes care of device specific stuff*/

        /* prompt for id number, 0 through 255 */
        printf("\nEnter any number between 0 and 255.\n");
        printf("This will be the unique magic code placed into the image:  ");
        scanf("%d",&our_code);
        while(our_code<1 || our_code>256){
                printf(" Between 0 and 255 please  ");
                scanf("%d",&our_code);
        }

        /* feed back the binary code which will be embedded in the image */
        printf("\nThe binary sequence ");
        for(i=0;i<BITS;i++){
                j = 128 >> i;
                if( our_code & j)printf("1");
                else printf("0");
        }
        printf(" (%d) will be embedded on the image\n",our_code);

        /* now generate the individual snow images  */
        get_snow_images();
        loop_visual();  /* this gives visual feedback on 'tolerable' noise level */

        printf("\nWe're now to the simulated suspect... \n");
        loop_simulation();

        close_everything();

    return(0);
}
```